Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization*

Takumi Okamoto†
okamoto@sbl.cl.nec.co.jp
C&C Research Laboratories,
NEC Corp.,Kawasaki 216, Japan

Jason Cong
cong@cs.ucla.edu
Dept. of Computer Science,
Univ. of California, Los Angeles, CA 90095

Abstract

This paper presents an efficient algorithm for buffered Steiner tree construction with wire sizing. Given a source *and n* sinks *of a signal net, with given* positions *and a* required arrival time *associated with each sink, the algorithm finds a Steiner tree with buffer insertion and wire sizing so that the required arrival time (or timing slack) at the source is maximized. The unique contribution of our algorithm is that it performs Steiner tree construction, buffer insertion, and wire sizing simultaneously with consideration of both critical delay and total capacitance minimization by combining the performance-driven A-tree construction and dynamic programming based buffer insertion and wire sizing, while tree construction and the other delay minimization techniques were carried out independently in the past. Experimental results show the effectiveness of our approach.*

1. Introduction

For timing optimization in lower level design of VLSI, buffer insertion (or fanout optimization), interconnect topology optimization, and wire sizing play important roles, and a number of algorithms has been proposed for these problems.

On fanout optimization problem, most of previous work focused on buffer tree construction in logic synthesis [1, 19, 18]. The timing model used during this stage mainly consists of gate delay and roughly approximated interconnect delay, which is assumed to be piecewise linear with the number of fanouts. When the wiring effect is dominant as in deep submicron designs, such a fanout-based model may be significantly inaccurate compared with the actual delay. Another problem with traditional approach is in area estimation. Typically, only the total gate area is optimized, and the wiring area and the routability are not considered. As a result, although the total gate area of the synthesized netlist is quite small, it may not result in small layout. Recently, fanout optimization algorithms using layout information have been proposed [20, 9, 6]. [20] and [9] presented algorithms for fanout tree generation based on alphabetic trees and minimum spanning trees, respectively. In [6], an efficient algorithm using dynamic programming was proposed for buffer insertion on a given Steiner tree.

On interconnect topology optimization problem, the analysis in [22] and [4] showed that as we reduce the device dimension, the *distributed* nature of the interconnect structure must be considered, and conventional algorithms for total wire capacitance minimization does not necessarily lead to the minimum interconnect delay. To cope with this situation, topology optimization algorithms for deep submicron VLSI design have been proposed [8].

Moreover, the wire sizing algorithms [4, 3, 17, 21] can achieve further interconnect delay minimization by optimally assigning different wire width to each wire segment. [10] has integrated wire sizing and power minimization with the buffer insertion algorithm in [6] under a more accurate delay model taking signal slew into account. It was also shown in [2, 13] that simultaneous driver/gate and wire sizing can achieve further delay reduction.

Although steady progress has been made in buffer insertion, Steiner tree construction, and wire sizing for delay minimization, and encouraging experimental results were reported, we believe that these steps need to be carried out simultaneously to construct *wire-sized buffered Steiner trees* directly. The independent approach for these steps often leads to sub-optimal designs due to the following reasons. In the case of buffer insertion followed by Steiner tree construction, the wiring delay and routability can not be estimated accurately in buffer insertion. In the case of Steiner tree construction followed by buffer insertion and wire sizing, a Steiner tree optimized for delay does not necessarily result in a minimum-delay wire-sized buffered Steiner tree. Recently, [7, 11, 14] explored the possibility of combining these steps. A simple greedy algorithm was used in [7] for tree construction with wire sizing. [14] and [11] used A-tree [4] and P-tree [12], respectively, for tree construction combined with the buffer insertion in [6].

This paper presents an efficient simultaneous algorithm for Steiner tree construction, buffer insertion, and wire sizing. The algorithm maximizes the required time at source in the tree with consideration of the total capacitance minimization by combining the performance-driven A-tree construction and dynamic programming based buffer insertion and wire sizing. Since all of these algorithms are based on a bottom-up approach from sinks, the combination is very reasonable. Experimental results show that our approach outperforms existing approaches by up to 16% in terms of the maximum delay in trees.

2. Delay Models and Problem Formulation

2.1. Delay Models

As in most previous work on interconnect layout optimization, we adopt the Elmore delay model [5] for interconnects and a commonly used RC model for buffers. For wire e, let l_e, w_e, c_e and r_e denote length, width, capacitance, and resistance, respectively. Further, let e_v denote the wire entering node v from its parent. We use the following models

*This work is partially supported by National Science Foundation Young Investigator Award MIP9357582.

†This work was done while the author was in UCLA as a visiting scholar.

for interconnects delay D_{wire} and buffer delay D_{buff}:

$$c_e = (c_a \cdot w_e + c_f) \cdot l_e, \quad r_e = r_0 \cdot l_e / w_e$$

$$D_{wire}(e_v) = r_{e_v} \cdot (\frac{c_{e_v}}{2} + c(T_v))$$

$$D_{buff}(b, c_l) = d_b + r_b \cdot c_l,$$

where c_a, c_f, and r_0 are area capacitance, fringing capacitance, and resistance for unit-width unit-length wire, respectively, T_v is the subtree rooted at v, $c(T_v)$ is the capacitance of *dc-connected subtree*[1] in T_v rooted at T_v's root, d_b and r_b are buffer b's intrinsic delay and output resistance, respectively, and c_l is the load on buffer b. When wire e is very long, we can divide e into a sequence of wires connected by degree-2 nodes to capture the distributed nature of the interconnect delay.

2.2. Problem Formulation

We use *required arrival time* and *total capacitance* as our optimization objective. The required arrival time at the root of tree T_v, denoted $q(T_v)$, is defined as follows:

$$q(T_v) = \min_{u \in sinks(T_v)} (q_u - delay(v, u)),$$

where q_u is the required arrival time of sink u, $sinks(T_v)$ is a set of sinks of tree T_v, and $delay(u, v)$ is delay from v to u defined by our delay models. The total capacitance of tree T_v, denoted $c_{total}(T_v)$, is defined as follows:

$$c_{total}(T_v) = \sum_{e \in T_v} c_e + \sum_{u \in buffers(T_v)} c_u + \sum_{u \in sinks(T_v)} c_u,$$

where $buffers(T_v)$ is a set of buffers in tree T_v and c_u is loading capacitance of buffer or sink u.

These measures are useful since the required time is a typical objective when optimizing the performance of combinational networks, and the total capacitance is closely related to the power dissipation. If we assume the signal arrives at the root of T at $t = 0$, the timing requirements are met when $q(T)$ is non-negative. The *wire-sized buffered Steiner tree problem* is stated as follows:

Given: A source s_0 and sinks $s_1, s_2, \ldots, s_n$ of a signal net S, with given positions and a required arrival time associated with s_i $(1 \le i \le n)$.

Find: Steiner tree T_{s_0} that connects S and has wire sized and buffers inserted.

Objective: Maximize $q(T_{s_0})$ with minimization of $c_{total}(T_{s_0})$ as the secondary objective[2].

For simplicity of presentation, we assume that only one type of buffer is considered for the buffer insertion, and signal polarity is neglected. We shall point out how to generalize our algorithm to handle multiple types of buffers and signal polarity at the end of Section 4.3.

3. Related Work

We briefly review the A-tree algorithm in [4] and the buffer insertion and wire sizing algorithm in [6, 10], which are basis of our proposed algorithm.

[1] "dc-connected" means "directly connected by wires".

[2] Other objectives, such as minimization of $c_{total}(T_{s_0})$ under the constraints of $q(T_{s_0}) \ge 0$, can also be handled by our algorithm.

3.1. A-tree Algorithm

In [4], it was shown that a routing tree which minimizes the Elmore delay upper bound in [16] can be achieved by minimizing a weighted combination of the objectives of the minimum Steiner tree, the shortest path tree, and the "quadratic minimum Steiner tree" (a tree that minimizes the summation of source-node path lengths, taken over *all* possible node locations). Therefore, a minimum-cost *rectilinear arborescence* (*A-tree*) formulated in [15] is of interest since it addresses all of these terms in the delay upper bound formulation at once.

Definition 1: *A rectilinear Steiner tree T is called an A-tree if every path connecting the source s_0 and any node p on the tree is a shortest path. An A-tree is optimal if its total wire length is minimum.*

In [4], an efficient algorithm based on bottom-up tree construction from the sinks (or iterative subtrees merge starting with the set of sinks as the initial set of subtrees) was proposed for optimal *A-trees*, which extends the algorithm in [15]. While [4] is using *safe moves* which cannot "worsen the sub-optimality" of an existing set of subtrees and *heuristic moves* that may not lead to an optimal solution (see [4] for formal definition) for the subtrees merge, we use only the heuristic moves in the A-tree algorithm for simplicity (essentially the algorithm in [15]). In most cases, [15] has similar performance as the A-tree algorithm in [4].

The algorithm works as follows: it maintains a set named $ROOT$ consisting of the roots of current subtrees which will be eventually merged to form a single Steiner tree; initially, $ROOT$ contains the roots of n trivial trees, each consisting of a single sink. The algorithm then iteratively merges a pair of roots at (x_1, y_1) and (x_2, y_2) such that the "merged" root to be placed at $(\min(x_1, x_2), \min(y_1, y_2))$ is as far from the source as possible (assuming that all sinks lie in the first quadrant with s_0 at the origin for simplicity). It terminates when $|ROOT| = 1$. Formal description for the algorithm is in Figure 1. It can be easily extended to the general case, where the sinks are distributed in all four quadrants.

Procedure Heuristic_Atree()
1: $ROOT \leftarrow \{s_i \mid 0 \le i \le n\}$;
2: **while** $|ROOT| > 1$ **do**
3: Find u, $w \in ROOT$ such that the sum $\min(u_x, w_x) + \min(u_y, w_y)$ is maximum;
4: $ROOT \leftarrow ROOT + \{v\} - \{u\} - \{w\}$, where v is a node with coordinates $(\min(u_x, w_x), \min(u_y, w_y))$;
5: Merge T_u and T_w to T_v adding edges from v to u and w, respectively;
end while;
end procedure

Figure 1. A-tree algorithm using heuristic move

3.2. Buffer Insertion and Wire Sizing

For given required arrival times at the sinks of a given Steiner tree, the buffer insertion algorithm in [6] chooses the buffering position on the tree such that the required arrival time at the source is maximized, where the delay is calculated with the definitions in Section 2. In [10], wire sizing and the total capacitance minimization are integrated with the algorithm. These algorithms assume that the topology of the routing tree (or Steiner tree) is given, as well as the possible positions for the buffer insertion and wire-width change, which are called *candidate points* hereafter.

The algorithms are based on the dynamic programming technique. Let T_i' denote T_i with e_i. A key point of the algorithms is that at each candidate point i, a set of triples $(q(T_i'), c_{total}(T_i'), c(T_i'))$ is computed and maintained for possible configuration of buffer insertion and wire sizing for T_i'. Figure 2 shows an example of an option at node i. Each triple is called an *option* and simply denoted by z or (q_z, p_z, c_z). The algorithm consists of two phases as follows. During the first phase the function *bottom_up()* in Figure 3 computes the *irredundant set* of all possible options[3], Z_v, for each node v (candidate point) in the tree in a bottom up manner[4]. After Z_0 is computed, the best option z is chosen from Z_0 such that $max_{z \in Z_0}(q_z - R_{gate} c_z)$ with the minimization of p_z as the secondary objective, where R_{gate} is the output resistance of the source gate. The second phase traces back the computations of the first phase that led to this option, and determines the buffer positions and wire width for each edge on the way.

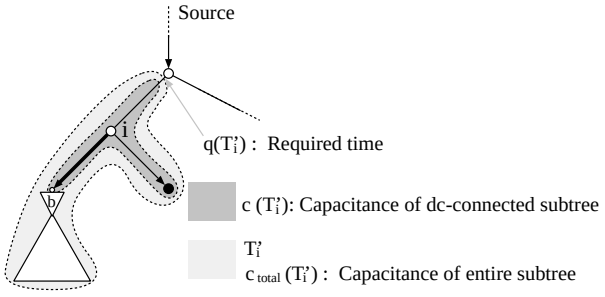


Figure 2. An option at node i

4. Simultaneous Steiner Tree Construction, Buffer Insertion, and Wire Sizing

4.1. Basic Idea of the Proposed Approach

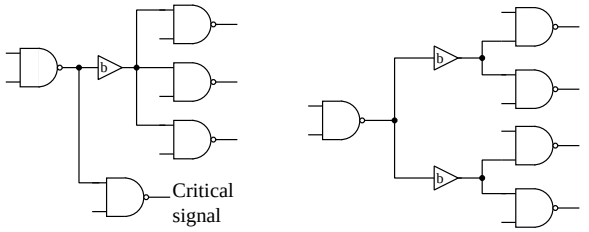
We develop an algorithm for simultaneous Steiner tree construction, buffer insertion, and wire sizing, called *wire-sized buffered A-tree* (*WBA-tree*) algorithm, by combining the algorithms in Section 3. This combination is very reasonable, since both of the algorithms in Section 3 are based on a bottom-up approach from the sinks. In the combination, the concepts of *critical path isolation* (Figure 4(a)) and *balanced load decomposition* (Figure 4(b)) are also applied, which are techniques used for fanout optimization in logic synthesis [1, 19, 18]. In logic synthesis, when one or several sinks are timing-critical, the critical path isolation technique generates a fanout tree so that the root gate drives only the critical sinks and a smaller additional load due to buffered non-critical paths. On the other hand, if required times at sinks are within a small range, balanced load decomposition is applied in order to reduce the load at the output of the root gate. Since these transformations are also applied recursively in a bottom-up manner from the sinks as the A-tree, buffer insertion, and wire sizing algorithms, it is natural to use these concepts in the combination.

[3] *Irredundant set* has no two options, (q, p, c) and (q', p', c'), such that $q > q'$, $p < p'$, and $c \leq c'$ [10].

[4] For simplicity, a binary tree is assumed here, but the algorithm is easily applied to general trees by addition of dummy nodes and 0 length wires [10]. Nodes which have only one child with either Z_u or Z_w being NULL in Figure 3, can also be handled by a simple extension.

```
Procedure bottom_up(T)
1: foreach v ∈ T in topological order from sinks to source do
2:   if v is a sink then
3:     Z'_v ← (q_v, c_v, c_v);
     else
4:     Z_u ← a set of options for v's left child;
5:     Z_w ← a set of options for v's right child;
6:     Z'_v ← irredundant merge of Z_u and Z_w;
7:     Z'_v ← Z'_v ∪ irredundant addition of buffer to
                  options in Z'_v;
       /* Addition of buffer b to option (q_z, p_z, c_z) results in
          (q_z − D_buff(b, c_z), p_z + c_b, c_b) */
     end if;
8:   Z_v ← ∅;
9:   for z ∈ Z'_v do
10:    foreach wire_width do
11:      Z_v ← Z_v ∪ (q_z − D_wire(e_v), p_z + c_{e_v}, c_z + c_{e_v});
         /*Wire sizing possibility for e_v*/
       endfor
     end for;
12:  Pruning of Z_v;
   end for;
end Procedure;
```

Figure 3. Algorithm for finding a set of options



(a) Critical signal isolation (b) Balanced load decomposition

Figure 4. Fanout optimization in logic synthesis

In our approach, the concepts of critical path isolation and balanced load decomposition are used when choosing two subtrees, T_u and T_w, to be merged to T_v in the A-tree algorithm, line 3 in Figure 1. Every pair of subtree roots u and w are evaluated based on the merging point v's location and the current required time at each subtree's root, v and w. Then, the best pair for the merge is chosen so that critical path isolation and balanced load decomposition are achieved with keeping the wire length as short as possible (See Section 4.2). For the evaluation, we keep a set of options at each of subtree's roots by using *bottom_up()* in Figure 3 during the construction of A-tree. Basically the following two steps are iterated in the WBA-tree algorithm.

1. Select u and w with taking critical path isolation, balanced load decomposition, and wire length minimization into account.
2. Merge T_u and T_w to T_v, and compute a set of options at v by *bottom_up*(T_v) in order to get the required time at v with considering buffer insertion and wire sizing.

In our implementation, the candidate points for buffer insertion are right after the Steiner points in the tree, which makes it possible to unload the critical path as much as

possible, and Steiner point itself for reducing the number of buffers inserted. Moreover, an edge whose length is longer than certain threshold given by user is divided recursively in order to make it possible to insert buffer in the middle of the wire (Figure 5). We also use the same set of points for the candidate points for wire-width change.

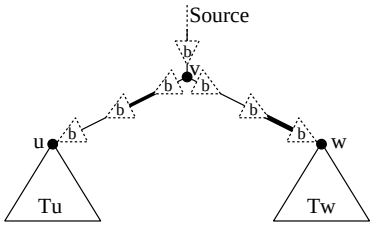


Figure 5. Candidate points for buffer insertion and wire-width change

4.2. Subtrees to be Merged in WBA-tree

In our algorithm, the computation of options, which is the first stage of the buffer insertion and wire sizing algorithm in Section 3, and the A-tree construction are performed simultaneously. Take the evaluation for the merge of T_u and T_w to T_v in Figure 6 as an example. Let Z_u and Z_w be the sets of options at u and w, respectively, computed in the previous steps. Since the parent nodes of the current subtree's roots, u and w, are not determined yet at this stage, l_{e_u} and l_{e_w} are assumed to be 0 in the computation of Z_u and Z_w. This means that lines 8-12 in Figure 3 are replaced by $Z_v \leftarrow Z'_v$. In order to evaluate this merge here, we *temporarily* compute Z_{vu} and Z_{vw}, which are the sets of options at u and w with assumption that the parents of u and w are v, respectively. The sequence of subtrees merge is based on the idea that merge of two subtrees where minimum value of the required times of the two subtrees is maximum among the current possible merging pairs leads to critical path isolation and balanced load decomposition.

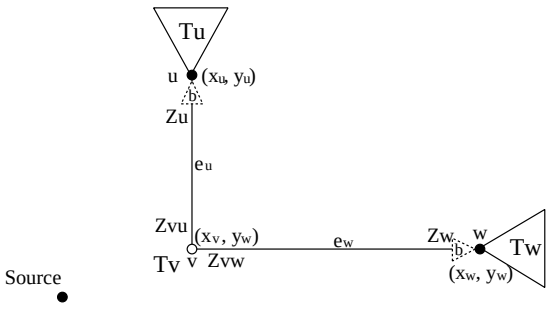


Figure 6. Evaluation for subtrees merge

We introduce the following definitions before describing how to select two subtrees to be merged in the WBA-tree construction.

Definition 2: The minimum value of the maximum possible $q(T'_u)$ and $q(T'_w)$, denoted Q_{uw}, is defined as follows:

$$Q_{uw} = \min\left(\max_{z \in Z_{vu}} q_z,\ \max_{z \in Z_{vw}} q_z\right),$$

where v is the merging point of T_u and T_w.

Definition 3: The maximum Q_{uw} among all possible merging pairs u and w in the set of roots, $ROOT$, for the current subtrees, denoted $Q_{max}(ROOT)$, is defined as follows:

$$Q_{max}(ROOT) = \max_{u,w \in ROOT} Q_{uw}.$$

Intuitively, we want to choose a pair of subtrees T_u and T_w to merge such that Q_{uw} is as large as possible. This helps to maximize the required time at the source.

Definition 4: The distance between the source and the merging point for u and w, denoted D_{uw}, is defined as follows:

$$D_{uw} = \min(u_x,\ w_x) + \min(u_y,\ w_y).$$

This definition is for the case that u and w are in the first quadrant with s_0 at the origin. Other cases can be defined in a similar way.

Definition 5: The maximum D_{uw} among all possible merging pairs u and w in the set of roots $ROOT$ of the current subtrees, denoted $D_{max}(ROOT)$, is defined as follows:

$$D_{max}(ROOT) = \max_{u,w \in ROOT} D_{uw}.$$

Recall that the heuristic A-tree algorithm always choose the pairs of subtrees T_u and T_w with $D_{uw} = D_{max}(ROOT)$ in order to maximize the possibility of subsequent merging with other subtrees to minimize the overall wire length. This has proven to be a very effective technique for wire length minimization [4, 15].

In order to consider both delay and total wire length minimization, we introduce the following definition of the *merging cost* for u and w.

Definition 6: The merging cost for u and w, denoted $mcost(u, w, ROOT)$, is defined as follows:

$$mcost(u, w, ROOT) = \alpha * \frac{Q_{uw}}{Q_{max}(ROOT)} + (1 - \alpha) * \frac{D_{uw}}{D_{max}(ROOT)},$$

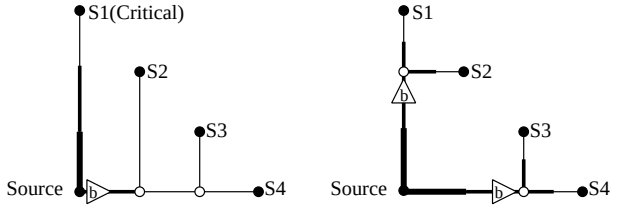
where α is a fixed constant with $0 \leq \alpha \leq 1.0$.

Note that instead of using $\alpha * Q_{uw} + \beta * D_{uw}$ for the cost, we use the scaled objective above. The use of the scaled objective avoids the problem of choosing a different pairs of α and β for each instance.

For the subtrees merge in the WBA-tree algorithm, we select T_u and T_w whose $mcost(u, w, ROOT)$ is maximum among all possible pairs of subtrees. Clearly, if we set $\alpha = 0$, our root selection criteria for the merge is the same as that in the A-tree algorithm presented in Section 3.1.

By using $mcost$ in the A-tree construction, required time maximization with buffer insertion (critical path isolation and balanced load decomposition) and wire length minimization can be achieved simultaneously. The first term contributes to the critical path isolation and balanced load decomposition as the fanout optimization in logic synthesis. The second term in $mcost$ contributes to the wire length minimization as the original A-tree algorithm. The effect of wire sizing can also be considered in the tree construction, since it is included in the computation of Q_{uw} through Z_{vu} and Z_{vw}. When one or several sinks are timing-critical,

these sinks are isolated since the merge for these sinks, whose Q values are smaller than the others, will be applied in the later stage. Figure 7(a) shows an example for this case, where sink s_1 is the most critical among all the sinks. Sink s_1 will be isolated since the merge with s_1 results in smaller Q values and will be applied after s_4, s_3, and s_2 are merged. On the other hand, if required times at sinks are within a small range, the merge will be performed so that the load is balanced, since Q of the merge for these sinks are also within a small range. Figure 7(b) shows an example for this case, where required times at sinks s_1, s_2, s_3, ands_4 are within a small range. The load will be balanced, since Q of the merge for the sinks are within a small range. The layout or geometric information is also taken into consideration due to the D term in the merging cost. For example, undesirable merge of s_1 with s_3 or s_2 with s_4 will not be used in general, as they result in much larger D values. Such information is not available to existing logic synthesis tools for fanout optimization. Note that, however, the resultant buffered trees are not necessarily planar.



(a) Tree with critical sink isolation

(b) Tree with balanced load decomposition

Figure 7. Example of WBA-tree

4.3. Summary of the WBA-tree Algorithm

The algorithm consists of two phases in the same way as the buffer insertion and wire sizing [6, 10]: the first phase is the bottom-up tree construction with option computation; the second phase is the top down buffer insertion and wire-width assignment. Formal description for the first phase is shown in Figure 8. Option computation at each subtree's root by *bottom_up()* at lines 3 and 8, and merging cost evaluation by $mcost(v, w, ROOT)$ at line 5 are integrated into A-tree algorithm. The second phase is the same with the one in the buffer insertion and wire sizing algorithm in [6, 10]. The option which gives the maximum required time and the minimum total capacitance at root is chosen, then traces back the computations of the first phase that led to this option. During the backtrace, the buffer positions and wire width for each segments are determined.

For the generalization of our algorithm, more than one types of buffers can be handled by computing options for each buffer type at line 7 in *bottom_up()*. Signal polarity also can be handled by adding signal polarity into the options and considering only the merge of options with the same polarity at line 6 in *bottom_up()* [10].

5. Experimental Results

We implemented the WBA-tree algorithm under the C/UNIX environment and used HSPICE to verify the results with accurate timing and power simulation. The technology parameters used in the experiments are based on the MCNC $0.5\mu m$ CMOS process model (R_{gate}: 270Ω, r_0: $0.12\Omega/\mu m$, c_a: $0.04fF/\mu m$, c_f: $0.15fF/\mu m$). We also used

Procedure WBA-tree_bottomup()
1: $ROOT \leftarrow \{s_i \mid 0 \le i \le n\}$;
2: **foreach** $v \in ROOT$ **do**
3: **bottom_up**(v);
/* Z_v is computed for each sink */
end for;
4: **while** $|ROOT| > 1$ **do**
5: Find $u,\ w \in ROOT$ with
$\max_{u,\ w \in ROOT} mcost(u, w, ROOT)$;
/* Z_{vu} and Z_{vw} are computed for the evaluation */
6: $ROOT \leftarrow ROOT + \{v\} - \{u\} - \{w\}$, where v is a node with coordinate $(\min(u_x,\ w_x), \min(u_y,\ w_y))$;
7: Merge T_u and T_w to T_v
adding edges from v to u and w, respectively;
8: **bottom_up**(T_v);
/* $Z_v|l_{e_v} = 0$ is computed with pruning,
$Z_u \leftarrow Z_{vu}$, and $Z_w \leftarrow Z_{vw}$*/
end while;
end procedure

Figure 8. Algorithm for simultaneous A-tree construction and option computation

the one type of buffer (r_b: 814Ω, d_b: $125ps$, c_b: $28fF$) and the set of wire width, $\{W_1, 2W_1, 3W_1\}$, where W_1 is the minimum unit width. The source in the tree is driven by an ideal voltage source and the input signal is a square wave with period of $40ns$ ($25MHz$).

For each net size, 20 nets were randomly generated on a $10mm \times 10mm$ routing region, and the average results were evaluated. The loading capacitances of the sinks were also randomly chosen from the interval $[0.05pF,\ 0.15pF]$. The required time at the sinks were set to 0 in order to evaluate the results with the maximum delay in the tree. We compared results obtained by the following four methods:

- T+B+W: Sequential A-tree construction [15], buffer insertion [6], and wire sizing [10].
- T+BW: A-tree construction [15] followed by simultaneous buffer insertion and wire sizing [10].
- TB+W: Simultaneous Steiner tree construction and buffer insertion (BA-tree [14] with $\alpha = 0.6$ in *mcost*) followed by wire sizing [10].
- TBW: Simultaneous Steiner tree construction, buffer insertion, and wire sizing (our WBA-tree construction with $\alpha = 0.6$ in *mcost*).

Table 1 shows the maximum delay, power dissipation, and wire length, where n, d, p, w denotes the number of sinks, maximum delay, power dissipation, and wire length, respectively. TBW reduced the maximum delay by 12-16% and 3-6% with the expense of 8% and 3% increase in power dissipation as compared with T+B+W and the other partially simultaneous methods (T+BW and TB+W), respectively. The runtime is less than 3 seconds for all data in T+B+W and TB+W, and about 3 seconds, 15 seconds, and 40 seconds for 10, 25, and 50 sinks, respectively, in T+BW and TBW (Sun SPARC 5).

Figure 9 shows the trade-off between delay, wire length, and power dissipation with α changed between 0 and 0.9 in TB+W and TBW for nets with 50 sinks. Note that TB+W and TBW with $\alpha = 0$ corresponds to T+B+W and T+BW, respectively. As for delay and wire length, smooth trade-off curves are obtained, while the power dissipation is decreasing with the delay at first and then increasing around

Table 1. Maximum Delay, Power Dissipation, and Wire Length

	T+B+W			T+BW			TB+W			TBW		
n	$d(ns)$	$p(mW)$	$w(mm)$	$d(ns)$	$p(mW)$	$w(mm)$	$d(ns)$	$p(mW)$	$w(mm)$	$d(ns)$	$p(mW)$	$w(mm)$
10	2.25	4.86	25.5	2.04	5.12	25.4	2.20	4.93	27.0	1.98	5.21	26.9
25	2.86	8.96	42.4	2.64	9.45	42.4	2.70	9.16	47.6	2.51	9.66	47.1
50	3.34	14.37	60.7	3.19	15.05	60.7	2.98	14.71	71.2	2.80	15.25	70.0

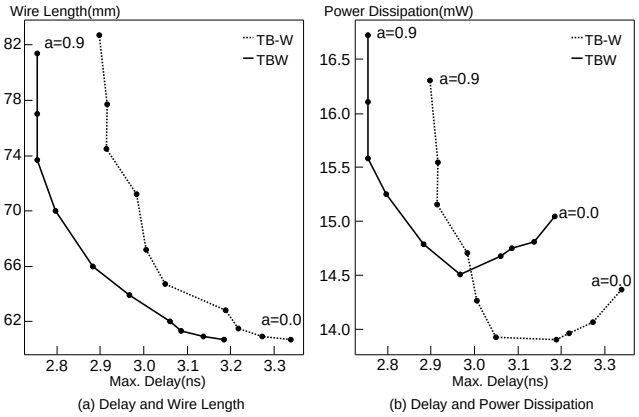


Figure 9. Trade-off between delay, wire length, and power dissipation

$\alpha = 0.4$. This kind of relationship can also be observed in transistor sizing.

6. Conclusions

In this paper, we presented the *WBA-tree* (*wire-sized buffered A-tree*) algorithm, which performs Steiner tree construction, buffer insertion, and wire sizing simultaneously with consideration of both critical delay and total capacitance minimization by combining the performance-driven A-tree construction and dynamic programming based buffer insertion and wire sizing algorithms. Experimental results showed that our approach outperforms existing approaches by up to 16% in terms of the maximum delay in trees.

References

[1] C. L. Berman, J. L. Carter, and K. F. Day, "The fanout problem: From theory to practice," *Advanced Research in VLSI: Proc. 1989 Decennial Caltech Conf.*, pp.69-99, 1989.

[2] J. Cong and C.-K. Koh, "Simultaneous Driver and Wire Sizing for Performance and Power Optimization," *IEEE Trans. VLSI*, 2(4), pp.408-423, Dec. 1994.

[3] J. Cong and K. S. Leung, "Optimal Wiresizing Under the Distributed Elmore Delay Model," *IEEE Trans. Computer-Aided Design*, 14(3), pp.321-336, Mar. 1995.

[4] J. Cong, K. S. Leung, and D. Zhou, "Performance-Driven Interconnect Design Based on Distributed RC Delay Mode," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp.606-611.

[5] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifier," *J. Applied Physics,* 19, pp. 55-63, 1948.

[6] L.P.P.P. van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp.865-868.

[7] T. D. Hodes, B. A. McCoy, and G. Robins, "Dynamically-Wiresized Elmore-Based Routing Constructions," *Proc. IEEE Int. Symp. Circuits Syst.*, 1994, pp.463-466.

[8] A. B. Kahng and G. Robins, *On Optimal Interconnection for VLSI*, Kluwer Academic Publishers, 1995.

[9] L. N. Kannan, P. R. Suaris, and H. G. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp.327-332.

[10] J. Lillis, C. K. Cheng, and T. T. Lin, "Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp.138-143.

[11] J. Lillis, C. K. Cheng, and T. T. Lin, "Simultaneous Routing and Buffer Insertion for High Performance Interconnect," *Proc. the Sixth Great Lakes Symp. on VLSI*, 1996

[12] J. Lillis, C. K. Cheng, T. T. Lin, and C. Y. Ho, "New Performance Driven Routing Techniques with Explicit Area/Delay Tradeoff and Simultaneous Wire Sizing," *Proc. ACM/IEEE Design Automation Conf.*, 1996, pp. 395-400.

[13] N. Menezes, R. Baldick, and L. T. Pileggi, "A Sequential Quadratic Programming Approach to Concurrent Gate and Wire Sizing," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp.144-151.

[14] T. Okamoto and J. Cong, "Interconnect Layout Optimization by Simultaneous Steiner Tree Construction and Buffer Insertion," *Proc. Fifth ACM/SIGDA Physical Design Workshop*, 1996, pp.1-6.

[15] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The Rectilinear Steiner Arborescence Problem," *Algorithmica* 7, pp.277-288, 1992.

[16] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. Computer-Aided Design*, 2(3), pp.202-211, 1983.

[17] S. S. Sapatnekar, "RC Interconnect Optimization under the Elmore Delay Model," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 387-391.

[18] K. J. Singh and A. Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem," *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp.357-360.

[19] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang, "Performance Oriented Technology Mapping," *Proc. sixth MIT VLSI Conf.*, pp.79-97, 1990.

[20] H. Vaishnav and M. Pedram, "Routability-Driven Fanout Optimization," *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp.230-235.

[21] T. Xue, and E. S. Kuh, "Post Routing Performance Optimization via Multi-Link Insertion and Non-Uniform Wiresizing," *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 575-580.

[22] D. Zhou, F. P. Preparata, and S. M. Kang, "Interconnection Delay in Very High-Speed VLSI," *IEEE Trans. Circuits Syst.*, 38(7), pp.779-790, July 1991.